

## GUI («Graphical User Interface») del 2

- Repetisjon
- Programmeringsparadigmer
- OO-strukturen av Swing
- Tilpasning av GUI-utseendet
- Tråder
- Tegning
- Tripp-trapp-tresko

Se også

- Big Java kapittel 10-11
- Programkoden i <https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/programmer/GUI/>

# Hvordan bygge opp et GUI-program

- 1 Finn ut nøyaktig hva programmet skal gjøre.
- 2 Lag en håndtegning på papir av alle elementene.
- 3 Skriv kode som genererer elementene.
- 4 Lag EventHandlerer-klassene som angir hva som skal skje når brukeren trykker på knappen.
- 5 Skriv resten av programmet.

## Demo 3: En teller

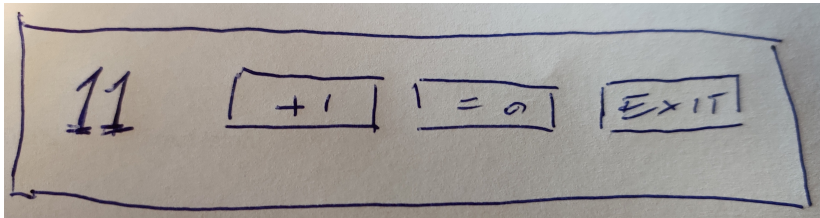
### Hva skal den gjøre?

- Det skal vise antall trykk.
- For hvert trykk skal antallet øke med 1.
- Det skal også finnes en knapp for nullstilling.



Del 1: Hva skal programmet gjøre?

## Del 2: En håndtegning



## Del 3: Kode som lager elementene

### Tellerverdien

```
JLabel antall = new JLabel(" 0 ");
```



Selve verdien lagrer vi i en int-variabel.

```
static int tellerverdi = 0;
```

## Telleren

```
 JButton tell = new JButton(" +1 ");
```



## Nullstilleren

```
 JButton resett = new JButton(" = 0 ");
```

## Avslutteren

```
 JButton slutt = new JButton("Exit");
```

## Del 4: Hva skjer ved museklikk?

### Telleren

```
class OekTeller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        ++tellerverdi;
        antall.setText(" " + tellerverdi + " ");
    }
}
tell.addActionListener(new OekTeller());
```



## Del 1: Hva skal programmet gjøre?

## Nullstilleren

```
class Nuller implements ActionListener {  
    @Override  
    public void actionPerformed (ActionEvent e) {  
        tellerverdi = 0;  
        antall.setText(" " + tellerverdi + " ");  
    }  
}  
reset.addActionListener(new Nuller());
```





## Avslutteren

```
class Stopper implements ActionListener {  
    @Override  
    public void actionPerformed (ActionEvent e) {  
        System.exit(0);  
    }  
}  
slutt.addActionListener(new Stopper());
```



# Hele programmet (del 1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Teller {
    static int tellerverdi = 0;

    public static void main (String[] arg) {
        JFrame vindu = new JFrame("Teller");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        vindu.add(panel);

        JLabel antall = new JLabel(" 0 ");

        JButton tell = new JButton(" +1 ");
        class OekTeller implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                ++tellerverdi;
                antall.setText(" " + tellerverdi + " ");
            }
        }
        tell.addActionListener(new OekTeller());
    }
}
```



## Hele programmet (del 2)

```
    JButton resett = new JButton(" = 0 ");
    class Nuller implements ActionListener {
        @Override
        public void actionPerformed (ActionEvent e) {
            tellerverdi = 0;
            antall.setText(" " + tellerverdi + " ");
        }
    }
    resett.addActionListener(new Nuller());

    JButton slutt = new JButton("Exit");
    class Stopper implements ActionListener {
        @Override
        public void actionPerformed (ActionEvent e) {
            System.exit(0);
        }
    }
    slutt.addActionListener(new Stopper());

    panel.add(antall); panel.add(tell); panel.add(resett);
    panel.add(slutt);

    vindu.pack();
    vindu.setVisible(true);
}
}
```



# Programmeringsparadigmer

Det finnes mange **programmeringsparadigmer**,<sup>1</sup> for eksempel

- **Imperativ programmering** der utførelsen følger programflyten angitt av *programmereren*.
  - **Objektorientert programmering** er en undergruppe der operasjonene er knyttet til objekter.
- **Hendelsesdrevet programmering** («event-driven programming») der *brukerens* handlinger styrer programflyten.

---

<sup>1</sup>Et programmeringsparadigme er måten vi tenker på når vi programmerer.

## Hendelsesdrevet programmering

Her ligger programmet passivt og venter på at noe skal skje. Dette *noe* kan være

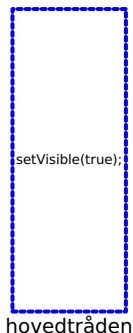
- brukeren trykker på en knapp på skjermen
- brukeren flytter musen
- brukeren trykker på en mustast
- brukeren trykker på en tast på tastaturet
- brukeren slipper opp en tast
- brukeren endrer størrelsen på vinduet
- et vindu kommer til syne fordi vinduet over fjernes

... og mye annet.

## Oppstart

Programmet startes i metoden main i hovedtråden.

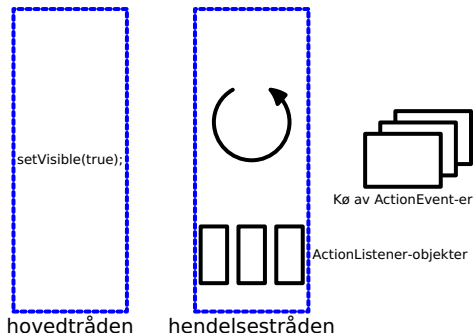
Etter en stund kaller programmet vårt `vindu.setVisible(true)`.



## Hva skjer i hendelsesdrevet programmering?

## Vente på hendelser

Det opprettes en ny tråd: hendelsestråden («event dispatch thread»). Så lenge køen er tom, ligger den passiv og venter på at noe skal skje.

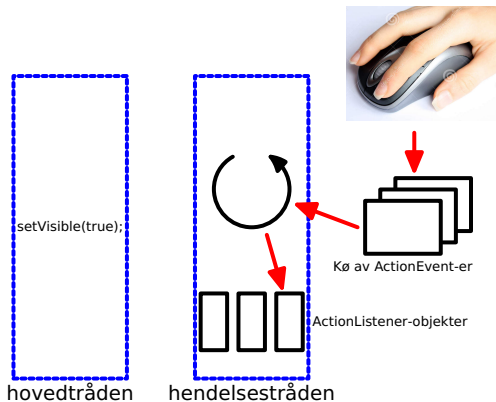


## Hva skjer i hendelsesdrevet programmering?

## Håndtere hendelser

Hver hendelse som inntreffer, resulterer i et `ActionEvent`-objekt i køen.

Hendelsesløykjen tar `ActionEvent`-ene etter tur, og den korrekte `ActionListener` vil bli kalt.





## Hvorfor trenger vi en kø av ActionEvent-er?

Hendelsestråden kan bare ta seg av én hendelse av gangen, men noen ganger kan flere hendelser inntreffe omtrent samtidig.

Da trenger vi køen for å ta vare på de hendelsene som venter på å bli tatt hånd om.

- Ingen hendelser må bli glemt.
- Hendelsene må håndteres i riktig rekkefølge.

## Hvilke tråder har vi?

Å ha to tråder innebærer følgende:

- De to trådene går uavhengig av hverandre.
- Hovedtråden kan fortsette med sine egne ting uten å forstyrre eller bli forstyrret av hendelsestråden.
- Hovedtråden kan kommunisere med hendelsestråden ved å lage egne `ActionEvent`-er; se dokumentasjonen til `SwingUtilities.invokeLater`.
- Selv om den ene tråden dør, lever den andre videre.

**NB!**

Det er derfor vi trenger

`Window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`.

- Kun én `ActionListener` kan jobbe av gangen; hvis den bruker lang tid, virker hendelseshåndteringen død.



Hvilke tråder har vi?

## Treg hendelsehåndtering

For å demonstrere effekten av treg håndtering av hendelser, kan vi legge inn en **venteløkke**:

```
class Nuller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        tellerverdi = 0;
        antall.setText(" " + tellerverdi);
        ventAktivt(10);
    }
}
```

En passende venteløkke kan defineres slik:

```
static void ventAktivt (int n) {
    // Kjør kode som tar omtrent n sekunder.
    for (int i = 1; i <= n; i++) {
        long x = 1;
        for (long j = 1; j <= 250*1000*1000; j++) {
            x = (3+j)*x % (1000*1000);
        }
    }
}
```



## Hvilke tråder har vi?

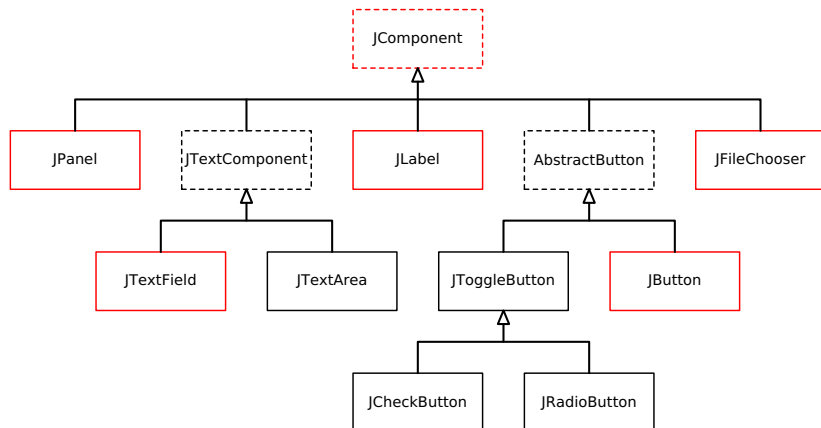
For moro skyld lar vi hovedtråen telle sekunder for å vise at den jobber uavhengig av hendelsestråden:

```
public static void main (String[] arg) {
    JFrame vindu = new JFrame("Treg teller");
    vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel();
    vindu.add(panel);

    :
    vindu.pack();
    vindu.setVisible(true);

    // Hovedtråen fortsetter på egen hånd:
    for (int sek = 0; sek < 30; ++sek) {
        System.out.print(sek + " ");
        ventAktivt(1);
    }
    System.out.println("ferdig");
}
```

# Swing er objektorientert

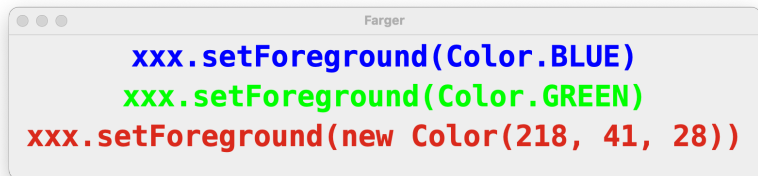


## Tilpasninger

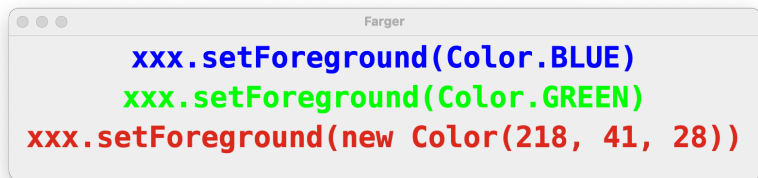
Siden mange GUI-egenskaper er definert i JComponent, finnes de i alle subclassene:

### Farger

Farger angis med `xxx.setForeground(...)` og `xxx.setBackground(...)`, for eksempel:



```
xxx.setForeground(Color.BLUE)
xxx.setForeground(Color.GREEN)
xxx.setForeground(new Color(218, 41, 28))
```



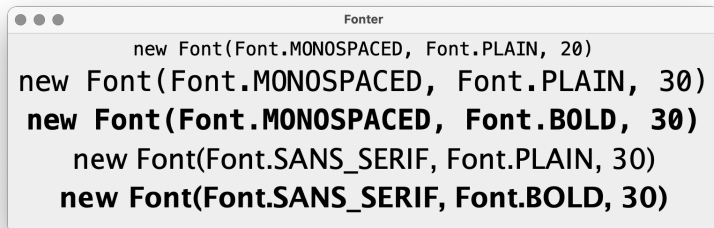
```
xxx.setForeground(Color.BLUE)
xxx.setForeground(Color.GREEN)
xxx.setForeground(new Color(218, 41, 28))
```

Farger lages med **new Color(r,g,b)** men noen farger er predefinert:

BLACK, BLUE, CYAN, DARK\_GRAY, GRAY, GREEN,  
LIGHT\_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

## Font

Fonten endres med `xxx.setFont(new Font(...))`.



```
new Font(Font.MONOSPACED, Font.PLAIN, 20)
new Font(Font.MONOSPACED, Font.PLAIN, 30)
new Font(Font.MONOSPACED, Font.BOLD, 30)
new Font(Font.SANS_SERIF, Font.PLAIN, 30)
new Font(Font.SANS_SERIF, Font.BOLD, 30)
```



## Rammer

Det finnes ca 15 rammer å velge blant med **setBorder(BorderFactory.createLineBorder(Color.BLACK))** og tilsvarende.



(I IN1010 vil vi kun bruke standardramme og LineBorder.)

## Tegning

Hittil har vi kun jobbet med bokser og tegneflater: JLabel, JButton og JPanel. Er det mulig å lage tegninger med linjer og sirkler også i GUI-vinduet?

Svaret er: Nei, ikke i Swing. Men vi kan få dette til med «lavnivå» mekanismer i AWT ved å lage vår egen JComponent:

```
class GPanel extends JComponent {
    GPanel () {
        setPreferredSize(new Dimension(300,300));
    }

    @Override
    public void paintComponent (Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
```

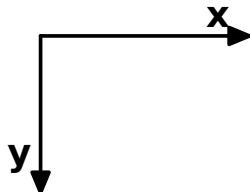
I Graphics2D har vi kommandoer for å tegne linjer, sirkler, polygoner etc i et koordinatsystem.

## Enhet

Enheten er alltid **pixler** (dvs skjerm punkter).

## Koordinatsystem

I koordinatsystemet går y-aksen *nedover*!



## Referansepunkt

Referansepunktet for sirkler, rektangler etc er alltid øvre venstre hjørne (selv for sirkler!).

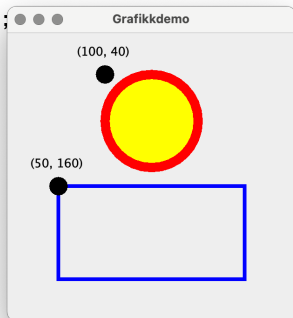
## Hvordan tegne med linjer, sirkler etc

```
class GPanel extends JComponent {
    GPanel () {
        setPreferredSize(new Dimension(300,300));
    }

    @Override
    public void paintComponent (Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;

        g2.setColor(Color.YELLOW);
        g2.fillOval(100,40, 100,100);
        g2.setColor(Color.RED);
        g2.setStroke(new BasicStroke(10));
        g2.drawOval(100,40, 100,100);

        g2.setColor(Color.BLUE);
        g2.setStroke(new BasicStroke(4));
        g2.drawRect(50,160, 200,100);
    }
}
```



Og, for ordens skyld, her er hovedprogrammet:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Grafikkdemo {
    public static void main (String[] arg) {
        JFrame vindu = new JFrame("Grafikkdemo");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        vindu.add(panel);

        panel.add(new GPanel());

        vindu.pack();
        vindu.setVisible(true);
    }
}
```

## Design av GUI-programmer

Det er to hovedmåter å organisere et GUI-program på:

- Lag det grafiske bildet først og fyll det etterpå med kode.
  - 👍 Ofte enklere å programmere.
  - 👍 Krever mindre kode.
- Utvikle «selve programmet» og GUI-presentasjonen hver for seg (MVC = Model-View-Controller)
  - 👍 Lettere å holde oversikten om «selve programmet» er komplisert.
  - 👍 Lettere å tilpasse programmet til ulike medier (f eks mobil, nettbrett og vanlig datamaskin).

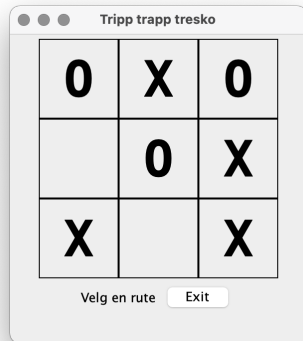
## Demo 4: Tripp-trapp-tresko

Vi skal lage TTT-spillet vist i forrige uke.

Vi skal gjøre det ved å bygge opp GUI-bildet og legge kode i det (alternativ 1 på forrige plansje).

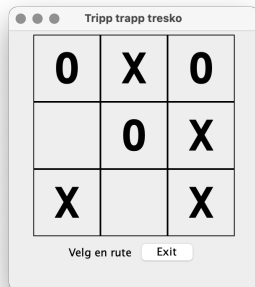
### Et tips fra egen erfaring

Det lønner seg å unngå GUI-kode i konstruktørene; i stedet bør den skilles ut og legges i en egen metode (f eks kalt `initGUI`).



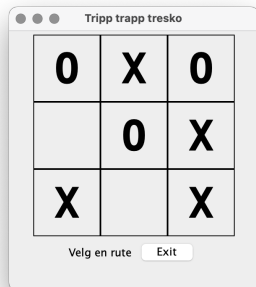
I vår løsning skal vi ha et vindu (**JFrame vindu**) med

- et spillebrett (**JPanel->Spillbrett Brett**) med
  - selve spillet (**JPanel->Spill spillet**) med rutenettet med
    - $3 \times 3$  ruter (**JButton->Rute[] ruter**)
  - et tekst med info til brukeren (**JLabel statustekst**)
  - en stoppknapp (**JButton sluttknapp**)





- Hver rute på  $3 \times 3$ -brettet skal være klikkbar.
- Ved hvert klikk skal programmet
  - 1 sjekke om trekket er lovlig
  - 2 vise brukerens trekk
  - 3 sjekke om brukeren har vunnet
  - 4 beregne og vise maskinens trekk
  - 5 sjekke om maskinen har vunnet



## Definisjon av de klikkbare rutene

Vi lager en egen subklasse Rute fordi vi ønsker å lagre et merke i hver rute. De lovlige verdiene skal være:

- ' ' når ruten er ledig
- 'X' når maskinen har valgt ruten
- '0' når spilleren har valgt ruten.

## Klassen Rute blir da slik:

```
class Rute extends JButton {
    Spill spillet;
    char merke = ' ';

    Rute (Spill s) {
        spillet = s;
    }

    void initGUI () {
        setBorder(BorderFactory.createLineBorder(Color.black));
        setFont(new Font("Monospaced", Font.BOLD, 50));
        setPreferredSize(new Dimension(80, 80));
        setText(" ");

        Rute denneRuten = this;
        class Rutevelger implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                if (! spillet.ferdig)
                    spillet.spill0(denneRuten);
            }
        }
        addActionListener(new Rutevelger());
    }

    void settMerke (char c) {
        setText(""+c); merke = c;
    }
}
```



## Spillebrettet er en subclasse av JPanel.

```
class Spillbrett extends JPanel {
    JButton sluttknapp;
    JLabel statustekst;
    Spill spillet;

    Spillbrett () {
        spillet = new Spill(this);
    }

    void initGUI () {
        spillet.initGUI();
        add(spillet);

        statustekst = new JLabel("Velg en rute");
        add(statustekst);

        sluttknapp = new JButton("Exit");
        class Stoppbehandler implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0);
            }
        }
        sluttknapp.addActionListener(new Stoppbehandler());
        add(sluttknapp);
    }
}
```



## Selve spillet er et $3 \times 3$ rutenett.

```
class Spill extends JPanel {
    Spillbrett brettet;
    Rute ruter[] = new Rute[9+1];
    boolean ferdig = false;

    Spill (Spillbrett b) {
        brettet = b;
        for (int i = 1; i <= 9; i++)
            ruter[i] = new Rute(this);
    }

    void initGUI () {
        setLayout(new GridLayout(3,3));

        for (int i = 1; i <= 9; i++) {
            ruter[i].initGUI();
            add(ruter[i]);
        }
    }
}
```

## Hovedprogrammet oppretter spillbrettet og starter det hele.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.util.Random;

class TTTGUI {
    public static void main (String[] args) {
        JFrame vindu = new JFrame("Tripp trapp tresko");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Spillbrett brett = new Spillbrett();
        brett.initGUI();
        vindu.add(brett);

        vindu.pack();
        vindu.setVisible(true);

        // Start spillet:
        brett.spillet.spillX();
    }
}
```

## Spilleren O (brukeren)

I klassen Spill finner vi koden som spiller det brukeren har valgt:

```
void spillO (Rute r) {
    if (r.merke != 'O') {
        brettet.statustekst.setText("Opptatt; velg en annen");
        return;
    }

    r.settMerke('O');
    if (harVunnet('O')) utropVinner('O');
    else if (erUavgjort()) utropUavgjort();

    if (!ferdig) {
        brettet.statustekst.setText("Velg en rute");
        spillX();
    }
}
```

## Spilleren X (maskinen)

I klassen Spill finner vi koden for maskinens spill:

```
Random tilfeldig = new Random();

void spillX () {
    int p;
    do {
        p = tilfeldig.nextInt(9)+1;
    } while (ruter[p].merke != ' ');
    ruter[p].settMerke('X');

    if (harVunnet('X')) utropVinner('X');
    else if (erUavgjort()) utropUavgjort();
}
```

(Hjelpemetodene harVunnet, erUavgjort, utropVinner og utropUavgjort finnes i koden på nettsiden.)



## Hvilke klasser og metoder trenger vi?

Disse klassene og metodene trenger man i IN1010. (Se også Big Java D.)

**class ActionEvent**`Object getSource()`**interface ActionListener**`void actionPerformed (ActionEvent e)`**class Border****class BorderFactory**`Border createLineBorder (Color c)`**class Color****class Dimension****class Font****class GridLayout****class LayoutManager****class JComponent**`void add (JComponent x)``void setBorder (Border b)``void setFont (Font f)``void setForeground (Color col)``void setPreferredSize (Dimension d)`**class JButton extends JComponent**`void addActionListener (ActionListener a)``void setText (String txt)`**class JFileChooser extends JComponent**`File getSelectedFile(JComponent c)``int showOpenDialog()`**class JFrame**`void add (JComponent x)``void pack ()``void setDefaultCloseOperation (int op)``void setVisible (boolean b)`**class JLabel extends JComponent**`void setText (String txt)`**class JPanel extends JComponent**`void add (JComponent x)``void setLayout (LayoutManager mgr)`**class JTextField extends JComponent**`void addActionListener (ActionListener a)``String getText()``void setText (String txt)`

Hvilke klasser og metoder trenger vi?

Til tegning i AWT:

**class BasicStroke implements Stroke**

**class Graphics**

**class Graphics2D extends Graphics**

void drawLine (int x1, int y1, int x2, int y2)

void drawOval (int x, int y, int width, int height);

void drawPolygon (int[] x, int[] y, int n)

void drawRect (int x, int y, int width, int height)

void drawString (String s, int x, int y)

void fillOval (int x, int y, int width, int height);

void fillPolygon (int[] x, int[] y, int n)

void fillRect (int x, int y, int width, int height)

void setColor (Color c)

void setStroke (Stroke s)

**class JComponent**

void paintComponent (Graphics g)

**interface Stroke**

